Restoring Security of Long-Living Systems by Co-Evolution

Jens Bürger*, Stefan Gärtner‡, Thomas Ruhroth*, Johannes Zweihoff*, Jan Jürjens*†, Kurt Schneider‡
*Chair of Software Engineering, TU Dortmund, Germany
{jens.buerger,thomas.ruhroth,johannes.zweihoff}@cs.tu-dortmund.de, http://jan.jurjens.de
†Fraunhofer ISST, Dortmund, Germany
‡Software Engineering Group, Leibniz Universität Hannover, Germany
{stefan.gaertner,kurt.schneider}@inf.uni-hannover.de

***Abstract*—[1]Security is an important quality aspect for modern information systems. Security properties may however be violated if the information system operates in an evolving environment. Environmental changes then trigger reactions which lead to co-evolution of the security design and the corresponding system model. However, updating the security design manually is time-consuming and error-prone. We present an approach to support semi-automatic system co-evolution which responds to environmental knowledge evolution, using the UML security extension UMLsec and graph transformation. The aim is to enable software engineers to react more reliably and effectively to environmental changes and to ensure lifelong compliance of information systems. To evaluate our approach, we conducted a case study on the open-source project iTrust.**

I. Introduction

Modern information systems are increasingly complex and need to operate in evolving environments. As a consequence, requirements to the system's security need to be revised and its model needs to be co-evolved to comply with the altered environment. The challenge is to determine changes of the system model based on environmental changes impacting security.

The approach presented in this paper to address this challenge is part of the *SecVolution* approach. It addresses long-living information systems with the goal to restore their security in face of evolving environmental knowledge and requirements [1]. On this account, required knowledge is modeled comprising common security and legal issues, used technologies as well as information about assets and access privileges [2]. To manage knowledge on different abstraction levels and its changes, layered ontologies and evolution operations are used [3].

In this paper, we present an approach for the semi-automatic co-evolution of design models in order to restore their security in presence of a changing environment. The goal is not to fully automate the process of security improvements in reaction to change, but to support software engineers as far as possible, so that they are able to focus their efforts on those security improvements which still have to be performed manually.

[1]This research is funded by the DFG project SecVolution (JU 2734/2-1 and SCHN 1072/4-1) which is part of the priority programme SPP 1593 "Design For Future - Managed Software Evolution".

To achieve this goal, the impact on the system model is determined by analyzing the evolved knowledge. Appropriate *reactions* are then inferred and applied to restore affected security properties semi-automatically. For this purpose, we need to investigate the following research questions:

- **RQ1**: *How to determine the impact of environmental changes relating to security on the system model of a given information system.*
- **RQ2**: *Whether a set of generic adaption rules exists that can be used, based on the derived impact, to co-evolve the system model and thus to restore affected security properties.*
- **RQ3**: *How to parameterize these generic rules in order to cope with a wide range of requirements of a concrete system model.*

The remainder of this paper is structured as follows. In Section II, we introduce background on SecVolution necessary for understanding the proposed co-evolution approach. In Section III, the approach is presented as well as an algorithm to co-evolve system models in detail. To evaluate our approach, we conduct a case study using the iTrust system with respect to privacy laws in Section IV. Related research from the field of security-related change analysis of design models is presented in Section V. Finally, we conclude our work and outline future research in Section VI.

II. Background on the SecVolution Approach

SecVolution [1] is built upon model-based software development using UML. To express security properties and assumptions in the model, we make use of the UML extension UMLsec [4] in which UML *stereotypes* and *tags* are used. Furthermore, *constraints* are used to assess if properties hold by the system design [5], [6].

Essential Security Requirements (ESR) are defined within SecVolution to model all security needs of the system that might be affected by environmental changes in any way. They are independent of a current technology and concrete algorithms (e.g. "keep account information secret from guest user" or "the data transfer needs to use a secure encryption"). In this paper, we use UMLsec to incorporate ESRs.

The knowledge required to refine ESRs for a specific information system within SecVolution is called *Security Context*

Knowledge (SCK). It includes but is not limited to attacker types and their abilities, exploited vulnerabilities, security obligations, and many others. Therefore, it introduces basic security concepts (e.g. *Entry Point*, *Trust Level*, and *Asset*) as well as domain-specific notions (e.g. *Personal Data*) and system-specific knowledge (e.g. *Patient Specific Instructions*). SCK usually evolves more often than the ESRs because knowledge about security techniques changes quickly, for example, due to newly discovered threats and vulnerabilities.

III. Approach to co-evolve system models

To check if the model under consideration undergoes a security violation, we analyze it using *model queries* which are realized using query languages such as the Object Constraint Language (OCL) or graph transformation specification languages.

The query is constructed such that it detects the non-compliance to given properties. We can formulate queries by a graph transformation and its underlying matching algorithm since models can be interpreted as graphs [6]. We additionally can make use of our tool platform CARiSMA [5] which supports analyzing models in UML (and other modeling notations) using approaches such as OCL to formalize the security properties under investigation.

Graph transformations can be used to formulate queries concerning static properties of a model. Queries are then modeled and evaluated using the established graph transformation framework Henshin [7]. Here, graph transformation rules are used where the left-hand side (LHS) equals the right-hand side (RHS). This means that the response to the query is a set of model elements which violate a given security property. Henshin has a unified view on the LHS and RHS of a transformation rule. In a nutshell, mappings between LHS and RHS are represented as so-called *actions*. These actions are annotated as stereotypes in a graphical representation. For further information about Henshin and how we use it to detect model vulnerabilities see [6].

To ease designing of graph transformation rules, Henshin enables interposing Java code between rule executions and *partial match*. This is a technique where specific nodes can be preallocated with actual elements of the instance model. Identifier of instance model elements can be obtained from previous and more general rules, which allows us to design rules with fewer nodes (cfer. [6]).

Sometimes a query needs to analyze the compliance of security annotations based on an information flow analysis or make use of automated theorem provers [8]. In this paper, we focus on queries using graph transformation and CARiSMA for risk and compliance checks.

A. Overview of the Approach

Figure 1 shows the components to realize the co-evolution algorithm. The knowledge changes ΔSCK of security context knowledge is the starting point. It shows how the environment of the regarded information system evolves. Potentially violated essential security requirements (ESR) are identified.

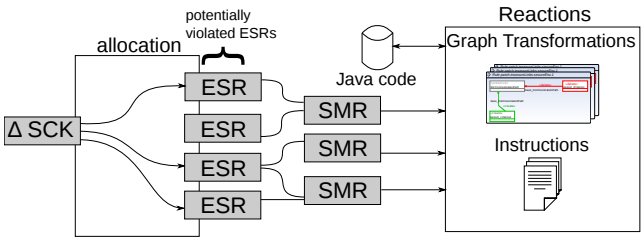

Fig. 1. Concepts used in the co-evolution approach and their relation.

To cope with these, the system model (SyM) need to be co-evolved accordingly which is realized by SMRs as provides by us. Every SMR treats preservation of one or more ESRs. On this, every SMR contains model queries to identify violating model elements. A sequence of co-evolution operations on the system model needs to be inferred from a sequence of evolution operations on the security knowledge. How the co-evolution approach presented in this paper is related to evolution and the SecVolution approach is explained in detail in [1]. An example of an ESR, its violation and countermeasures is given in Section IV-B1. SMRs further infer reactions to correct the violations which can be textual or using graph transformation. Reactions can also be supported by Java code as discussed in [6]. As a common base, our approach as well as supporting tools and approaches are built around EMF [9].

In Figure 2, we show the structure of security maintenance rules.

A security maintenance rule uses changes observed in the security knowledge and provides a set of corresponding changes to apply to the system model.Furthermore, a *pre-condition* ensures that a security maintenance rule is only applied to a system model that principally is annotated with the respective security property (cfer. [1]). For example, a security maintenance rule can treat the violation of a certain UMLsec security property. A number of queries is part of one SMR and can be used to parameterize the co-evolution function.

To *react* to the security violation discovered, model elements as obtained by the queries can be used. Furthermore, a query is related to a number of reactions. A reaction consists of a sequence of steps. Every reaction step has input and output parameters and can be either a direct manipulation of the system model by using graph transformation or a set of instructions for security experts to realize changes manually. For complex operations that cannot be easily expressed using graph transformation, we can additionally use Java code as

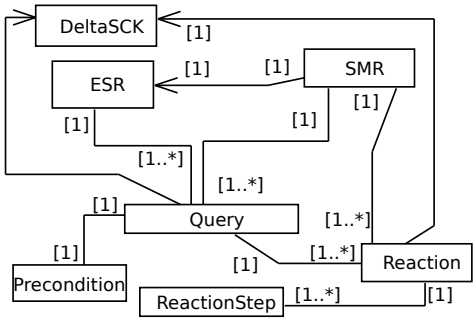

Fig. 2. Overview of the security maintenance rule and its components.

part of security maintenance rules. The execution order as well as parameter mapping is defined by the reaction (cfer. *preparation code* in [6]).

B. Semiautomatic Co-Evolution: An Algorithm

The proposed algorithm determines which model elements violate ESRs and, if necessary, recovers the compliance to the ESRs by semi-automatically applying appropriate reactions. As SecVolution security of long-living systems that were designed secure initially, we require that at a certain point of time the system model fulfills all security requirements with respect to the modeled SCK.

1) Query violating model elements: The information in ΔSCK is used to determine which ESRs are potentially violated. The violation type and thus a list of candidate ESRs are derived from certain classes of knowledge evolution as introduced in [3]. The next step is to check the model against every candidate ESR respecting the changes of the knowledge. Here, *check* means to query the model to find elements that violate security properties. This can be done by using e.g. CARiSMA or an query language as discussed above.

Every query as part of a security maintenance rule with a fulfilled precondition is evaluated. Thus, if the response to a query is empty, we know that the model does not undergo violations of the respective security property. Otherwise, the result is a set of model elements that point to precise spots in the model violating the security property.

2) Infer Co-Evolutions: The *reactions* part of the security maintenance rules incorporates the set of violating model elements from the previous activity. As there can be multiple reactions to treat a given maintenance object or query, every reaction is checked with respect to its applicability. For example, imagine a system where some data is not allowed to be transferred unencrypted anymore. Two possible reactions to this violation are (1) to encrypt the whole communication between the affected nodes or (2) to encrypt the critical data itself. Depending on the specific system, some reactions are applicable and others are not. According to the example, reaction (1) cannot be applied if the system under consideration is not a distributed system and thus the communication between different nodes is not part of the system model. On this account, to infer possible co-evolutions, information about the violating model elements as well as the security property under consideration is used.

3) Apply Co-Evolutions: To actually realize co-evolutions, SMRs need to be *applied* which means that reactions linked to the query are carried out at the system model. The first kind of reaction is to alter the system model directly. We perform analysis and alteration of system models using graph transformation [6], [7]. In some cases it may not be possible or necessary to apply a co-evolution in terms of model transformations. This is the case when the violation of an ESR does not demand changes to the system model but requires changes to the system's data. Thus, the security expert can be assigned to perform appropriate actions. We call this type of reaction *Instructions*.

If there is only one reaction applicable for a given security maintenance rule, it can be chosen automatically. Otherwise, a security expert is presented the alternatives and needs to choose the security maintenance rule that is suitable in the given context. In summary, the following situations can occur: (1) The reaction is determined fully automatically, (2) the reaction requires some user input or (3) the user needs to manually adapt the system.

Side-Effects and User Assistance: The algorithm terminates if no further ESRs is potentially violated. Otherwise, the security expert may manually check the results and do manual adaptations and eventually re-run the algorithm. This especially comes into account if two security requirements are to be applied that (logically) contradict each other (such as non-repudiation and anonymity).

IV. Application to a medical information system

To evaluate our approach, we perform a case study using the open-source system *iTrust*. Within the case study, we focus on privacy due to its importance for information systems where individual or personal data are possibly at stake. Nowadays, most of the information we rely on is digitalized to grant convenient access. Thus, information must be protect from unauthorized access.

According to our research questions, we investigate the impact of changes regarding the German Federal Data Protection Act (BDSG) [10] on the system model of the iTrust system (RQ1). Based on the derived impact, we evaluate whether an appropriate adaptation rule can be chosen to co-evolve the iTrust system model properly (RQ2). On this account, we parameterize these rules with respect to the needs of the concrete system model (RQ3). Finally, we discuss whether our approach is sufficient for maintaining security of long-living information systems.

A. Case Study Design

The role-based medical information system *iTrust* is designed to provide patients with medical information and to help medical staff organizing their daily work. The iTrust project was founded at North Carolina State University and is currently maintained by the Realsearch Research Group. It is a fully operational system, whose development artifacts, e.g. requirements and code, are publicly available.

In this case study, we focus on the use cases 3, 26, and 44. Use case 3 is of interest because it describes user authentication and thus resembles an entry point to the system as well as authentication of different roles. A user in the system can be a patient or a staff member. Use cases 26 and 44 describe access to and modification of patient-specific (private) data including instructions to the patient and diagnoses. Due to space restrictions, for further information we refer to the iTrust documentation [11]. The iTrust project does not feature design models. Since our approach makes use of models (i.e. state chart diagrams and class diagrams), we reverse engineered them based on the given code (cfer. Figure 4). We modeled the roles having access to certain states using the UMLsec

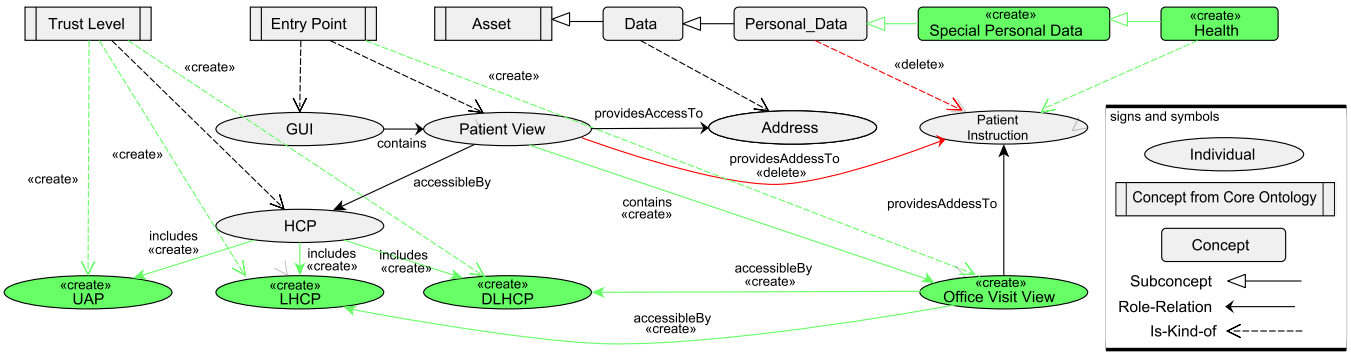

Fig. 3. Illustration of the knowledge to reflect the SCK changes based on the different versions of the BDSG. The concepts define the taxonomy of the BDSG according to [3]. The individuals represent the mapping between the taxonomy and the knowledge of the iTrust system.

stereotype ⟨⟨ensureRole⟩⟩. This stereotype has a tagged value `role` denoting that a given transition can only be activated if the user is assigned to this role.

1) Privacy Knowledge Modeling: In the case study, we analyze iTrust to ensure privacy properties with respect to the European and German privacy protection rules. Within the case study, we modeled relevant parts of the knowledge of the directive and the BDSG using ontologies [3]. To determine whether the initial models are compliant to the privacy and security regulations from 1990 as introduced above, we performed a security analysis on iTrust.

2) Privacy Knowledge Changes: Privacy regulations change regularly to reflect changes in the behavior and technical possibilities to work with data as well as new juristic interpretations and the German privacy protection act has been altered several times to fulfill the European directive [12].

We used the BDSG from 1990 as initial version. We compared it to the 2001 version which extends the notion of personal data by *special categories of personal data* such as data about racial or ethnic origin, political opinions, health and sex life, etc. (cfer. [13]). The access to this kind of data needs to be more restrictive as enforced in sec. 13 par. 2 BDSG. As iTrust processes different sort of health care data, access in iTrust must be restricted to retain compliant with the 2001 version of the BDSG.

In Figure 3, the regarded knowledge changes are modeled. Concepts and individuals in gray are valid for both BDSG versions (1990 and 2001), while the green elements have been added (in 2001) and the red elements have been removed.

B. Evaluation and Results

We assessed initial and co-evolved iTrust models using CARiSMA [5] to validate the case study. As a result, the initial models are compliant to the privacy and security regulations defined in the EU directive and the 1990 version of the BDSG. After co-evolution according to our approach has been performed, the iTrust model should be secure according to the 2001 version of the BDSG.

Based on the delta knowledge presented in Figure 3, the system model must be checked if a forbidden information flow exists in the system endangering given privacy requirements (ESR). If this is the case, the system model needs to be co-evolved to prevent this information flow.

1) Non-compliant Information Flows: From the knowledge changes according to the BDSG it can be inferred that certain users are not allowed to write or even read special categories of personal data. Based on this, a query belonging to a security maintenance rule (SMR) is selected and parameterized. The selected query is used to find those transitions in the state chart which violate the information flow. As parameters, the role `HCP` and the asset `PatientInstruction` are used.

Starting from the state `NotificationPageHCPHome`, which is only allowed for the HCP role to reach (see annotation ⟨⟨ensureRole⟩⟩ with {`role=HCP`}), the transition `OpenPatientInstructionDialog` has access to the data class `PatientInstruction`. It is derived based on use case 44 that the HCP has access to patient specific instructions (`PatientInstruction`). Additionally, it is inferred that the UAP is not allowed to access patient specific instructions after the BDSG evolved. As the HCP includes the UAP role, the access to patient specific instructions is considered a non-compliant information flow according to the 2001 version of the BDSG. Thus, a violation has been found in the system model.

2) Co-evolution of the iTrust System Model: To mitigate/fix this violation, the system model has to be co-evolved as described in Section III-B. For this purpose, corresponding reactions (part of the security maintenance rule) must be performed.

In the case study, the state `ModifyOV` is copied for the UAP role (`ModifyOV-UAP`) and the violated transitions and their corresponding states are deleted. Furthermore, the annotation for the transition to state `NotificationPageHCPHome` is changed to ⟨⟨ensureRole⟩⟩ with {`role=LHCP`}. Additionally, a new state named `NotificationPageUAPHome` and a corresponding transition to the copy of the state `ModifyOV-UAP` is added. In the remainder, we show how the reactions used for the case study are realized in more detail.

The security maintenance rule triggers the respective actions and model transformations.

First, the state `ModifyOV` is copied. To create a copy of the given object as well as all containing objects, helper methods of EMF [9] are used. This leads to a copy of state `ModifyOV` with a new name. Moreover, all transitions pointing to or from the original state are copied also. Source and target references

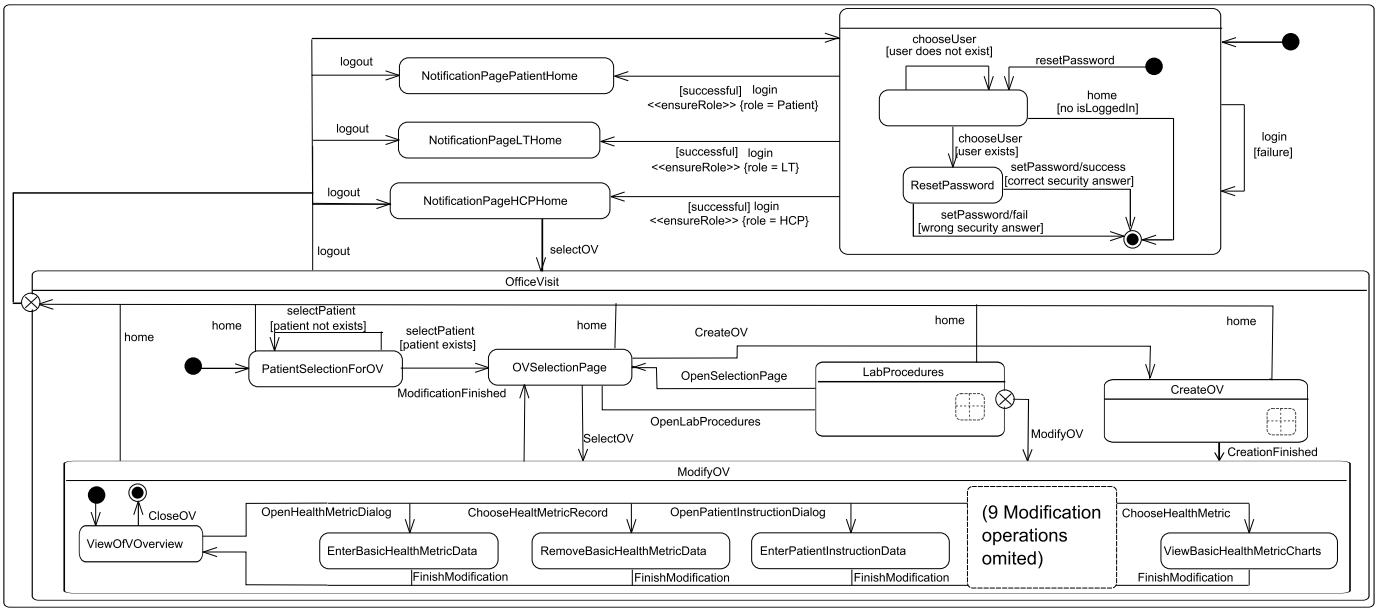

Fig. 4. State chart for use cases 3, 26, and 44.

are adjusted.

Next, the transitions pointing to `ModifyOV` and `ModifyOV-UAP` need to be enhanced by ⟨⟨`ensureRole`⟩⟩ to ensure that only the appropriate roles can access the respective states.

Figure 5 shows a graph transformation that is used to add the stereotype ⟨⟨`ensureRole`⟩⟩ as well as its tagged value to a given transition. ⟨⟨`preserve`⟩⟩ means that an element is member of the LHS as well as the RHS. ⟨⟨`create`⟩⟩ means that an element is only member of the RHS. ⟨⟨`delete`⟩⟩ means that an element is only member of the LHS. The transition can be identified either by its name (using the respective String parameter) or by an (EMF) object reference, for example obtained from the query (e.g. with partial match).

The graph transformation in Figure 6 is used to remove a given Transition. The Transition to be removed is given via an object reference (with partial match). All transitions pointing from the state are removed, too.

A state `NotificationPageUAPHome` is finally needed to split the role responsibility at the beginning of the discovered path. This works analogously to the previous steps using graph transformations and is omitted due to space restrictions.

The reaction part of the SMRs as presented above is realized

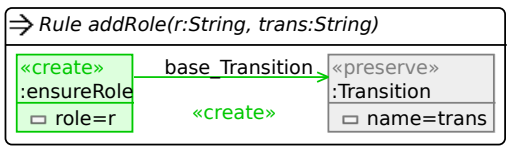

Fig. 5. Graph transformation adding ⟨⟨`ensureRole`⟩⟩ to transition.

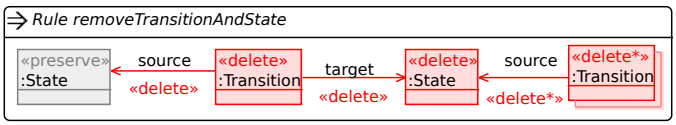

Fig. 6. Graph transformation removing a given transition and state.

in a number of single steps. As reaction steps have input and output data, a sequence of reaction steps is to be executed and assigns the respective parameter and return values to reach a certain goal (e.g. refine role access to certain transitions in a state chart). We already realized triggering a sequence of action in our previous work [6].

C. Discussion

We applied our approach to iTrust. As security knowledge base, we referred to the German Federal Data Protection Act (BDSG) from 1990 which iTrust was compliant with. To consider knowledge evolution, we compared the system to the 2001 version of the BDSG. It now appears that iTrust is not compliant due to more severe regulations and needs to be co-evolved. Regarding ***RQ1***, the impact of the knowledge change on the system model is determined with model queries. As result, the impact type and corresponding information assets are derived. The impact on the system model relies on the information contained in the given state charts (behavior) and class diagrams (structure). Thus, the level of detail of these artifacts has a large influence on the quality of the results. As we assume model-based development, these details must be provided to generate executable code.

Regarding ***RQ2***, we introduced *security maintenance rules* to mitigate the information flow violation in this case study. We provide appropriate model transformations enriched with Java code encoding independent basic operations such as copying and removing states/transitions as well as assigning roles to transitions. Thus, they can also be applied to other systems from different domains. A minor restriction is that the transformations presented are applicable for state charts only.

With regard to ***RQ3***, the necessary co-evolution is inferred. The SMR and its components are generic in a sense that they are independent from a specific model. They can be applied

to a given system via parameterization. For this purpose, the information retrieved from model query is used as parameters.

In this case study, we only focused on information flow based violations. For this type of violation, we have shown that our approach enables semiautomatic co-evolution of development models compared to manual assessment and adaptation. Considering other security principles, further research is needed.

D. Threats to Validity

A qualitative case study of a single software product such as iTrust does not necessarily lead to a generalizable conclusion. The selection of iTrust for the case study is based on the facts that we needed an information system that has to change caused by changes in privacy laws. As iTrust does not provide any design models, we reverse engineered them based on the code. We are aware of the threat of a manipulated case study caused by this task. To eliminate this threat, the reverse engineering has been done by an independent team who was not involved in the development of our approach. Moreover, we decided to use class diagrams as well as state charts which are common development artifacts.

V. Related Work

Our approach combines methods and concepts from the fields of security impact analysis of design models and security knowledge modeling.

In [14], a pattern-based approach is presented to cope with co-evolution. A pattern encodes the relationship between several artifacts to capture and handle changes systematically. To adapt corresponding software architecture, a change specification is analyzed in order to select appropriate patterns. The work in [15] focuses on the security impact of software enhancements. Therefore, security requirements patterns to identify threats are combined with security design patterns to determine effective countermeasures. The overall goal is to enable engineers to estimate and compare the amount of modifications needed by different countermeasures. In [16], an approach is proposed that uses regular expression-based patterns to identify vulnerabilities in software architecture. Moreover, the authors show in their evaluation that their approach can be used by engineers who are not experts in security. In contrast to the related pattern-based approaches, our co-evolution approach can cope with a wide variety of design models due to using UMLsec and graph transformations.

In [17], a rule-based impact analysis approach for heterogeneous software artifacts is introduced. The goal of the analysis is to identify impacted artifacts and to determine how they are affected. To guide software engineers through the ongoing change process, [18] uses change impact knowledge retrieved over time. To realize effective change management, the approach in [19] uses semantic knowledge about artifacts and change operations organized in an ontology. Change operations are formalized as graph rewriting rules encoding the change and its impact propagation.

VI. Conclusion and Future Work

In this paper, we presented an approach to co-evolve design models of long-living information systems. This is supported by formalized security knowledge that is subject to constant change. Our approach makes use of knowledge deltas and analyzes the impact on the system by means of model queries. Necessary co-evolution steps are then inferred and applied semi-automatically using graph transformation, supported by textual instructions and code. In the case study iTrust, we have shown the applicability of our approach.

There are some research tasks left for future work: Our approach is not free of side-effects. Thus, ESRs may lead to logically contradictory security properties or there may be co-evolution steps that interfere in a more implicit way. It needs to be investigated how these interfering co-evolutions can be detected and resolved in advance. This can for example be examined in a user study, observing how often co-evolutions are manually reverted.

References

[1] J. Bürger, J. Jürjens, T. Ruhroth, S. Gärtner, and K. Schneider, "Model-based security engineering with UML: Managed co-evolution of security knowledge and software models," in *Foundations of Security Analysis and Design VII: FOSAD Tutorial Lectures*, ser. LNCS, vol. 8604, 2014, pp. 34–53.

[2] S. Gärtner, T. Ruhroth, J. Bürger, K. Schneider, and J. Jürjens, "Maintaining requirements for long-living software systems by incorporating security knowledge," in *22nd RE Conf.* IEEE, 2014, pp. 103–112.

[3] T. Ruhroth, S. Gärtner, J. Bürger, J. Jürjens, and K. Schneider, "Towards adaptation and evolution of domain-specific knowledge for maintaining secure systems," in *PROFES 2014*, ser. LNCS.

[4] J. Jürjens, *Secure Systems Development with UML.* Springer, 2005.

[5] "CARiSMA tool homepage," 2013, http://carisma.umlsec.de/.

[6] J. Bürger, J. Jürjens, and S. Wenzel, "Restoring security of evolving software models using graph-transformation," *Int. Journal on STTT*, 2015, Springer Online First. [Online]. Available: http://link.springer.com/article/10.1007/s10009-014-0364-8

[7] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: Advanced concepts and tools for in-place EMF model transformations," in *MoDELS 2010*, pp. 121–135.

[8] J. Jürjens, "Sound methods and effective tools for model-based security engineering with UML," in *ICSE 2005*. ACM, pp. 322–331.

[9] Eclipse Foundation. Eclipse modeling framework project (EMF). [Online]. Available: http://eclipse.org/modeling/emf/

[10] BMI, "Bundesdatenschutzgesetz," Bundesgesetzblatt, 2005.

[11] L. Williams, A. Meneely, S. Smith, L. Hayward, B. Smith, and J. King. iTrust electronic health care system. [Online]. Available: http://agile.csc.ncsu.edu/iTrust

[12] E. Parliament, "Directive 95/46/EC," *Official Journal of the European Union*, vol. L 281, pp. 0031–0050, 1995.

[13] BfDI, "BDSG Änderungen," 2014. [Online]. Available: http://www.bfdi.bund.de/bfdi_wiki/index.php/BDSG_%C3%84nderungen

[14] K. Yskout, R. Scandariato, and W. Joosen, "Change patterns," *SoSyM*, vol. 13, no. 2, pp. 625–648, 2012.

[15] T. Okubo, H. Kaiya, and N. Yoshioka, "Analyzing impacts on software enhancement caused by security design alternatives with patterns," *Int. Journal of Secure Software Engineering*, vol. 3, no. 1, pp. 37–61, 2012.

[16] M. Gegick and L. Williams, "On the design of more secure software-intensive systems by use of attack patterns," *INFSOF*, vol. 49, no. 4, pp. 381–397, 2007.

[17] S. Lehnert, Q. Farooq, and M. Riebisch, "Rule-based impact analysis for heterogeneous software artifacts," *CSMR 2013*, pp. 209–218.

[18] J. Diaz, J. Perez, J. Garbajosa, and A. Yague, "Change-impact driven agile architecting," in *HICSS 2013*, pp. 4780–4789.

[19] M. Bouneffa and A. Ahmad, "The change impact analysis in BPM based software applications: A graph rewriting and ontology based approach," in *Enterprise Information Systems*. Springer, 2014, pp. 280–295.